BCG | SCBX

The Agent is the Easy Part

Sepember 2026

By Tjun Tang (BCG), Luc Grimond (BCG), Srihari Chakarajan (BCG), Amir Alsbih (BCG), Dennis Trawnitschek (SCBX), Suttipong Kanakakorn (SCBX)

Image generated with AI

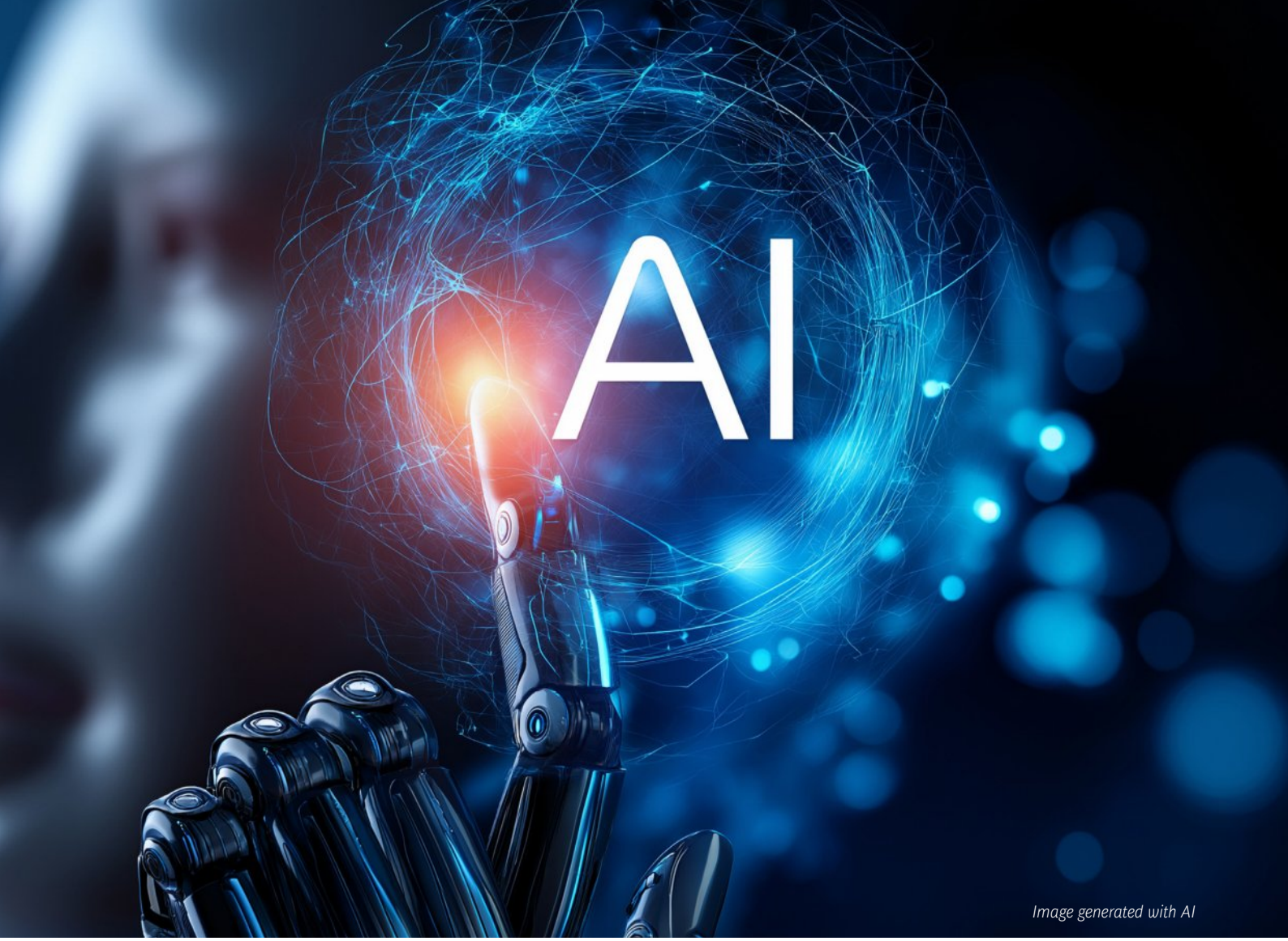


Image generated with AI

Executive Summary

The AI model stopped being the bottleneck a while ago. The system that has to stay trustworthy around it did not. What follows is a reference architecture for agentic banking, the decisions a bank actually has to make, and a clear account of the point of difference that defines whether you end up with a production system or another demo.

Agentic AI in banking requires a clear allocation of authority. Models can interpret language, assemble context, and propose a course of action. The bank still needs explicit rules for access, approval, execution, and evidence. The following seven sections set out the design choices that matter in production:

1. **Start with the gap and ask the right question**
2. **Put the architecture in one picture**
3. **Follow one call from end to end**
4. **Scale safely: the agent proposes; the system commits**
5. **Protect the economics: keep routine work away from the model**
6. **Make the management decisions before scaling**
7. **Design for the difficult cases**

1. Start with the gap and the right question

We went looking for the business case for agentic AI in our own data and found it in a pattern that has nothing to do with AI. Across several thousand collections and servicing calls, first-call resolution varied sharply between our strongest and weakest performers. The calls used comparable scripts and covered similar situations, yet the outcomes varied sharply.

That spread is the opportunity. Labor savings are part of the business case, but they usually follow a change in the work mix: routine contacts move to automation, while people handle exceptions and sensitive cases. The immediate operational benefit is more consistent resolution across customers, channels, and times of day. At scale, that consistency must translate into lower cost, additional capacity, or both.

Most banks never get there because they spend their energy on the wrong question.

The question in most steering committees is whether the model can be trusted. That is a distraction. Evaluation can measure and bound what a model does on a defined task, but it cannot turn a probabilistic system into a deterministic one.

The useful question is: where does the probabilistic part end, and where does the deterministic part take over?

Put that way, agentic AI in banking stops looking like an intelligence problem and starts looking like a delegation problem. A bank does not need a smarter agent. It needs a system that knows exactly what the agent is allowed to see, say, recommend, trigger, or execute, and under what conditions. That is a different thing to build than the demo most teams start with. In a regulated industry, a single error such as a mistaken phone number can cause significant damage to the bank and trigger regulatory fines.

The cost of getting this wrong is now visible. Gartner expects more than 40% of agentic AI projects to be canceled by the end of 2027, citing runaway cost, unclear value, and weak controls. A 2025 MIT study of enterprise GenAI found only about 5% of integrated pilots producing measurable profit impact. On WebArena, a widely cited long-horizon benchmark, the strongest GPT-4 agent of that period finished approximately one task in seven where people finished three in four. Most of these projects do not fail because the model was too weak. They fail because nothing serious was built around it.

Image generated with AI

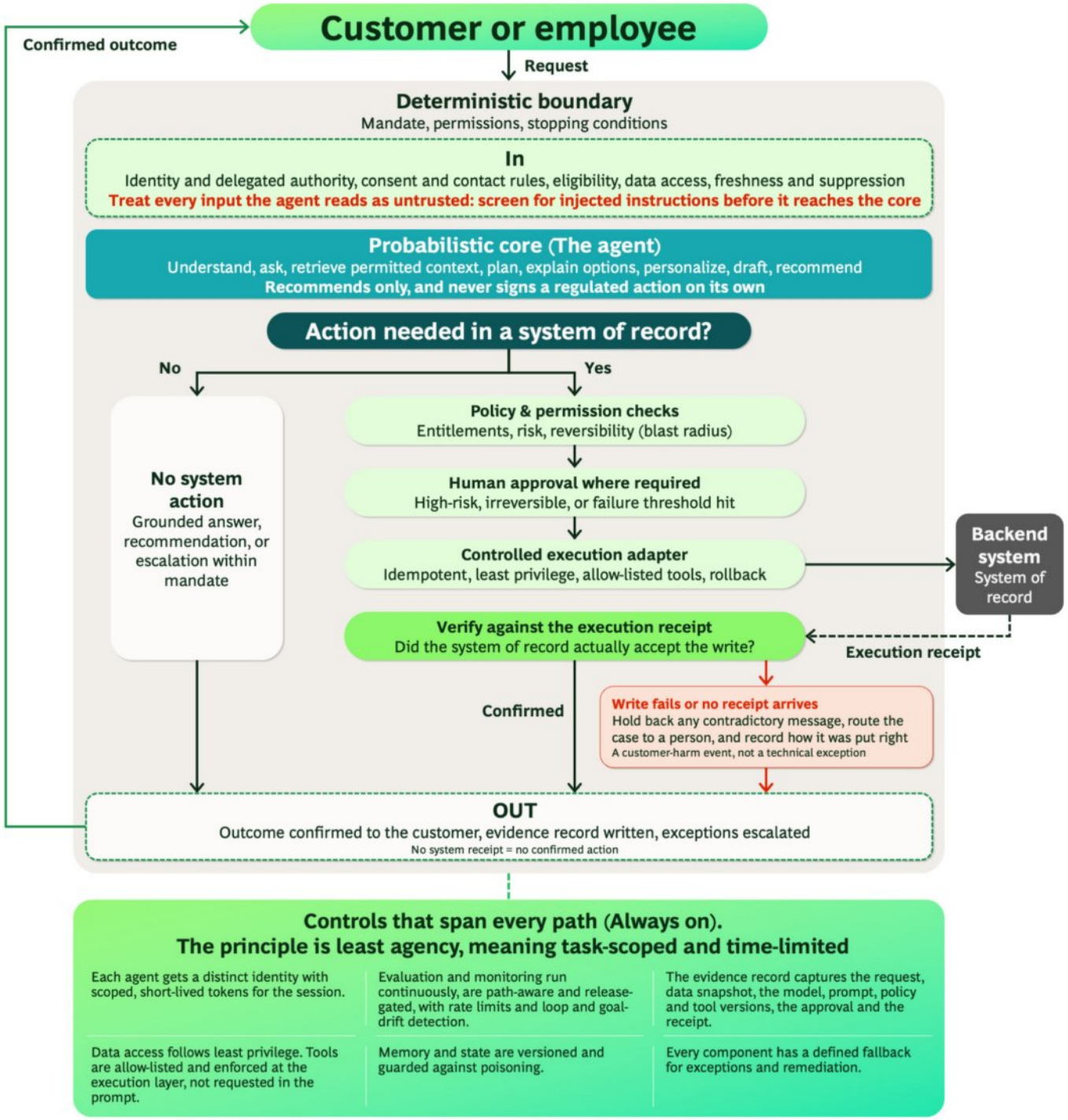
2. Put the architecture in one picture

Strip away the vendor diagrams and the pattern that survives in production is simple to state. A probabilistic core sits inside a deterministic boundary. The model reasons, talks, plans, and proposes inside that boundary. Deterministic code decides what is allowed, commits what is approved, verifies the result, and records what happened. [Exhibit 1.]

EXHIBIT 1

A probabilistic core inside a deterministic boundary

The model proposes, policy decides, a controlled adapter commits, the system verifies against a receipt, and the evidence record proves it.



This means the agent recommends only, policy decides, a human approves where the stakes call for it, a controlled adapter executes, the system confirms the result, and an audit trail proves it. Under this structure, a (non-deterministic) model never signs a regulated action on its own.

The confirmation step is the one people skip, and it is the one that hurts. An agent that tells a customer 'your payment plan is set' before the core system has accepted the write has already done the damage, whether or not the write later goes through. The rule is clear: the agent never reports a result it cannot prove. No system receipt means no confirmed action.

What you keep as evidence is the structured decision and the inputs behind it, not the model's raw private reasoning. Stored chain-of-thought is closer to a liability than to proof, and a regulator wants to see which rule was applied at which version with what data, not what the model briefly considered.

The boundary does not script every path. It defines the mandate, the permissions, and the stopping conditions. Inside it, the agent still reasons freely. Outside it, the bank stays in control.

3. Follow one call from end to end

Imagine a customer whose salary is late. Within its strict operational boundary, the AI acts like a skilled human agent. It verifies their identity, explains the debt, interprets their tone, and negotiates a payment from savings before fees accumulate. Today's models handle this conversational work exceptionally well.

What it cannot touch is the commitment. The payment plan it can offer comes from a small, approved set. The promise-to-pay it records is checked against the live account before it counts. And it physically cannot dial a customer who has already paid, filed a complaint, or been flagged for hardship, because those are rules in the boundary rather than lines in a prompt.

Now change one crucial fact. The customer's mother is in the hospital, they broke their last promise, and the date they can manage falls outside policy. This is the call that tells you whether you have a system or a demo. The core still negotiates with empathy, but the boundary routes the case to the right hardship path and, where the rules require it, hands the customer to a person with the full context already gathered.

The cooperative call is the easy part. The complex and compassionate consideration is the critical piece the whole architecture has to be designed for.

4. Scale safely: the agent proposes; the system commits

In a regulated business the instinct is to read the deterministic boundary as a brake. It is closer to the opposite. The only reason you can point an agent at all of your customers is that it cannot promise the wrong thing or contact the wrong person. Take the boundary away and you have not built a faster bank. You have built a faster way to mis-sell.

Putting mandatory rules in the execution layer can prevent the agent from initiating actions outside the permitted schema. It does not remove failures caused by incorrect data, defective code, integration errors, or weak policy. Performance should therefore be measured against the existing human process, with separate tests for rule compliance, customer outcomes, and operational reliability.

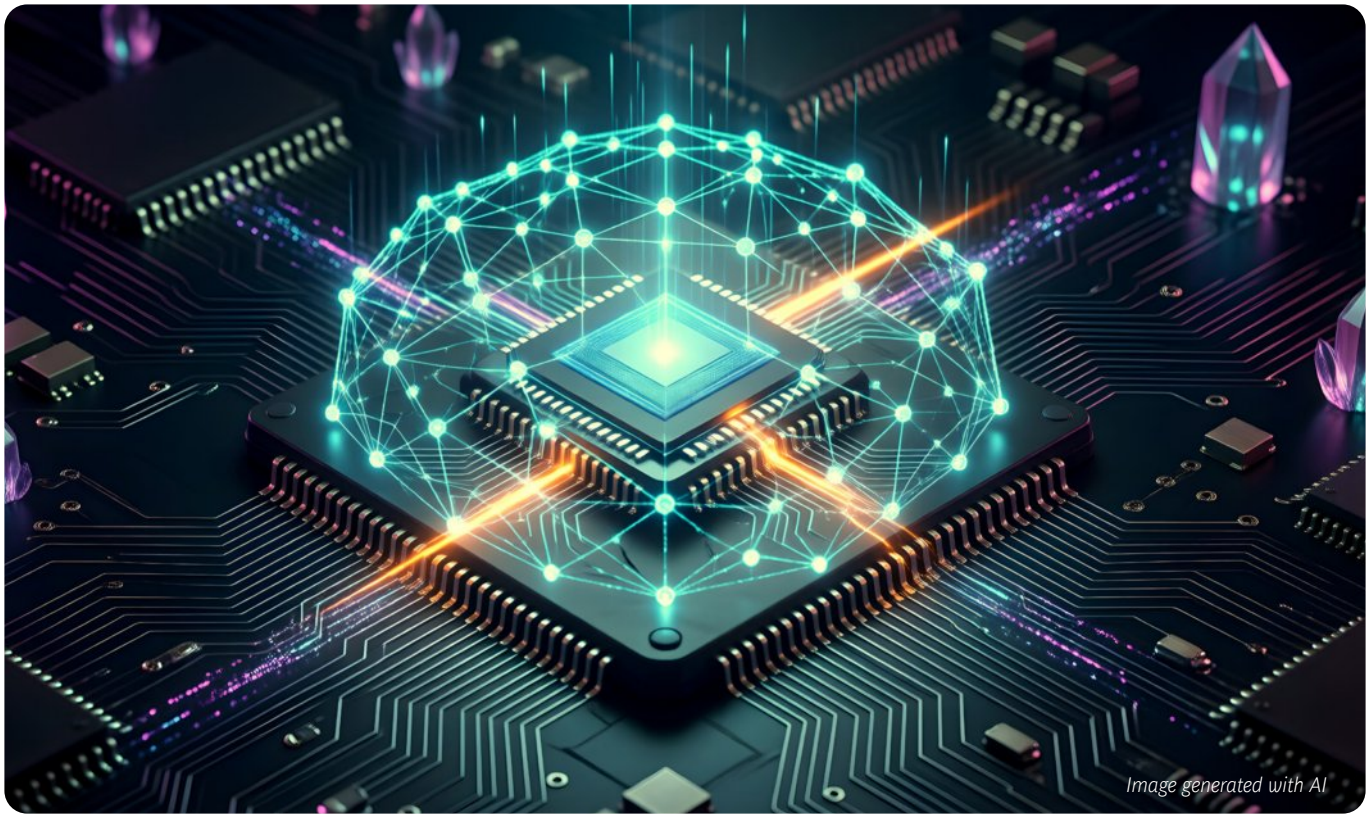
The common mistake runs in the other direction. A team catches the agent doing something wrong and responds by prompting harder, stacking 'NEVER do X' in capitals into the instructions. It does not hold, because a model does not weigh emphasis the way a person does. A rule the bank cannot break belongs in versioned, auditable code, not in a strongly worded paragraph. One strong rule of thumb to internalize: if you can draw the logic as a flowchart, write it as code.

This is also why it pays to classify every action by how much damage it can do and how easily it can be undone. Reading a record is not the same as drafting a message. Drafting a message is not the same as contacting a customer. Contacting a customer is not the same as changing a repayment plan, which is not the same as moving money.

Reversible, low-stakes actions can run on their own from the first day. Irreversible, customer-facing ones start as 'agent proposes, human commits' and earn more freedom only as the evidence comes in.

The question underneath all of it is how far the action of a single agent travels before something deterministic, or someone human, stops it. That distance, the blast radius, is the risk number that matters. Because in this business it is the single worst outcome, not the average, which must be considered.


Image generated with AI



5. Protect the economics: keep routine work away from the model

The second design choice determines the cost base. When we mapped the documented procedures across a banking group, most steps fell into one of three broad groups. Some are simple operations, such as retrieving a field, checking whether a document is present, or updating a status. Others apply a written rule to known inputs. A smaller group require open-ended judgment or a difficult customer conversation.

Implementation should follow the same division. Workflow automation is suitable for simple operations. Rules or conventional models can handle decisions with defined inputs and outcomes. A language model should be used where language, ambiguity, or interaction makes it useful.

Sending all these actions through a large model because the model can handle it is how programs quietly run out of money. A model may still be the wrong tool. Each model call adds cost, latency, and output variability to a step that may require only a fixed rule or a conventional workflow. It's worth reinforcing that an agent is not one 'call' action. It is a loop that resends a growing context at every step. In Anthropic's research system, agent runs used about four times as many tokens as chat interactions, while multi-agent runs used about 15 times as many. These figures are specific to that architecture, but they show why multi-agent designs should be reserved for work whose value justifies the additional context, coordination, and tool use. For most banking workflows a single, well-bounded agent is both cheaper and steadier.

The true discipline then is to match the engine to the task. Rules and ordinary machine learning for toil and decidable judgment, the model reserved for the steps where reasoning pays for itself. A small number of reusable agent patterns covered most of those procedures, which tells you where the leverage sits. A lot of the highest-value automation is not agentic at all. It is plain deterministic automation that finally got an intelligent front door.

The same logic runs down to which model you pick. Most workflow steps do not require the strongest available model. Model selection should be based on an explicit quality threshold for each task, with routine work routed to a smaller or cheaper model when it meets that threshold. Published RouteLLM evaluations reported cost reductions of up to 85% while retaining approximately 95% of GPT-4 performance on selected benchmarks. These are benchmark results rather than a production guarantee: actual savings will depend on the workload, model prices, routing accuracy, and quality threshold.

Build provider-agnostic, lead with small models, and route each step to the cheapest one that clears the bar, using a signal you already have. This signal could be as simple as 'summarize this transcript' versus 'decide the next collections action', rather than guessing difficulty from the text. Treat the whole thing as a fixed budget rather than a per-token rate, and cost stops being what ends the project.

6. Make the management decisions before scaling

The architecture of the right solution is not rigidly defined by a diagram. It is settled by a handful of choices that are management decisions, not technical footnotes. The architecture is determined by five management decisions:

- Who is accountable for each decision?
- Which decisions must remain deterministic?
- Which components should the bank own, and which should it rent?
- Which controls should be centralized, and which choices should remain local?
- How fresh must each data signal be?

Who owns each decision. The failure that quietly sinks these programs is collapsing everything into one idea of 'the AI decides'. It does not, and it must not.

There are five kinds of decision in play: (1) credit and eligibility, (2) personalization, (3) conversation, (4) operations, and (5) execution. Each needs exactly one accountable owner, and for most of them the owner is not a language model. Credit and eligibility belong to scoring models and policy. Whether a write is allowed belongs to deterministic code. The conversational layer owns how to say something and what to propose, and nothing more. If that ownership map is not locked before you build, every later correction is expensive and every regulatory question is unanswerable.

What stays deterministic. The agent can read a situation, choose a tone, suggest a path, and assemble context for a person. It should not be the thing that decides eligibility, suppression, contact policy, a fee-waiver limit, settlement authority, or whether a write is permitted. The test from the last section applies again. If a rule must never break, it does not live in the prompt.

What you own and what you rent. Labs ship new building blocks every few weeks and the platform vendors are constantly moving. That means building all of it yourself is a way to fall behind, and buying all of it is a way to lose control.

Draw the line by rate of change and by what you answer to a regulator for. Rent the parts that commoditize fastest: runtime, model access, developer tooling, generic orchestration. Own the parts you cannot delegate: identity, permissions, policy authority, the evaluation gate, and the evidence record.

You can buy the policy tooling. You cannot buy the policy authority. This is what we have come to call the orchestration tax: the layer you let someone else own is the one that prices everything above it, and the hardest to unwind later.

What you centralize and what you leave local. In a group with many subsidiaries this is the hardest decision for your operating model. Centralize the controls that create risk when they differ across the group: identity, permission patterns, the action taxonomy, the model gateway, the evaluation harness, evidence standards. Leave local the things that are a source of difference: channel experience, tone, treatment strategy, the journeys themselves, thresholds within policy.

Centralized does not mean one size fits all, and federated does not mean every subsidiary rebuilds the stack. The cost of getting this wrong is duplicated controls and fragmented evidence, and it should be priced in, not discovered later.

How fresh the data has to be. Running everything in real time is expensive and usually unnecessary. Running the wrong thing on stale data is dangerous. The decision should be judged as a maximum acceptable staleness per signal, set by use case.

Collections make the stakes obvious. The check for whether a customer has already paid, opened a dispute, or asked not to be contacted has to run fresh in the moment before contact, not from last night's batch, or the agent calls someone who settled an hour ago. For any regulated action, capture proof of the state at the moment it executed, not the state you assumed when the task began.

Once those decisions are made the architecture becomes much less mysterious. The bank does not need a giant agent platform first. It needs one production-grade slice where the boundary, the action path, the evaluation gate, and the evidence record work end-to-end, and the integration into the existing landscape is clear cut and managed.

Image generated with AI

7. Design for the difficult cases

The important question of agents at scale is one of effort, not technology. In normal software, most of the work happens before launch. With agents it inverts, and most of the work starts after go-live. Teams that treat launch as the finish line are the ones still stuck in pilot a year later. The ones that pull ahead can review what the agent did, find the cause, and fix it fastest, because the speed of that loop is what earns the confidence to expand.

Most of that work is evaluation, and the first thing to get right is what you measure. Recent trajectory-aware testing has shown that a large share of agent 'successes' are corrupt: the right outcome reached through a policy breach, an ungrounded claim, or a confirmation the agent invented. An accuracy dashboard marks all of those green.

For a collections agent, a case closed by promising something the bank never authorized is not a win. It is a liability you cannot see on a standard report. So measure the path, not just the answer, and measure it across many runs rather than one. The same task that passes most of the time on a single attempt will often fail at least once when it is repeated.

What works in practice is layered. Cheap deterministic checks on format and tool calls run on every code change. A model-as-judge handles qualitative review at scale, calibrated against known-bad traces rather than trusted on its own. Human review covers the edge cases.

The system sits behind a release gate with explicit thresholds for precision, hallucination, latency, and cost. Because these are defined by use case, any regression blocks the build just like a failing test. And the gate has to be loaded with the cases that actually break things: the disputed debt, the hardship, the customer who already paid, the write-back that fails, the agent that hits its budget mid-task. If the only thing you can test is the happy path, you are not ready to ship.

Systems must also fail safely with standard reliability engineering. Because AI agents automatically retry tasks, a blind retry on a partially finished step can easily turn a timed-out payment into a double charge.

The non-negotiables are familiar from any serious payments system: (1) idempotency keys on anything that moves money or sends a letter; (2) durable state for anything that runs longer than a single request; (3) loop detection because a task that closed in three steps last week can quietly spin into 60 calls tonight; and (4) a defined fallback for every component rather than a bare kill switch, since a kill switch with nothing behind it just turns one outage into a worse one.

The case that catches teams out is the write that fails after the customer has already been told. If the agent has said 'your plan is accepted' and the core then rejects the write because of a lock or a stale balance, the bank has a customer-harm event, not a technical exception. The system has to notice, hold back any contradictory follow-up, route the case to a person, and record how it was put right. Building that path is thankless. Not building it is a conduct risk.

Human oversight belongs in the architecture. But the useful question is not whether a human is in the loop. It is how you run oversight across an operation that handles hundreds of thousands of contacts. That means tiers, not a single approval button: reversible in-policy actions run with monitoring and sampling, higher-risk actions wait for approval, and the most sensitive ones (hardship, disputes, irreversible money movement) go to a person by design.

It also means the reviewer gets an independent case packet, assembled from the record, not the agent's own account of what it did, because a reviewer reading the agent's narrative tends to rubber-stamp it. Decide the dull questions up front: which queue an exception lands in, what the service level agreement (SLA) is, what happens when a step that should take four minutes takes four hours, and how the decision feeds back into the next evaluation. Oversight is an operating model, not a checkbox, and it is usually what decides whether the business case survives contact with scale.

Identity is the part most banks have not worked through, and it is the next attack surface. Allowing an AI to use permanent user credentials invites unrestricted security risks. The pattern now emerging gives each agent its own identity with scoped, short-lived tokens: this agent, acting for this user, with these permissions, for this session, all logged.

Knowing who an agent is does not cap what it does, and a kill switch on the token will not stop the calls already in flight, so identity has to be paired with rate limits and with actions designed to be reversible. Treat the tools as hostile territory too. Controlled studies have shown that tool interfaces become an attack surface in their own right, which is why the tools an agent can reach should be allow-listed and the permissions enforced at the execution layer, not requested in the prompt.

One idea is worth resisting outright: the agent that learns from every run and re-tunes itself in production. In a regulated book, quiet self-improvement is just quiet drift. Learning belongs offline: versioned, evaluated, and shipped through change control, with every model or policy update treated the way you would treat any dependency upgrade, tested against your baselines, staged, and reversible. A bank that lets its agents rewrite themselves will not be able to explain, six months on, why a decision changed.

What this adds up to

Agentic banking depends on the controls around the model. Before scaling, a bank should have:

- A deterministic boundary for regulated decisions and actions
- One accountable owner for each decision
- An appropriate engine for each type of work
- Evaluation of the path taken, not only the final answer
- Human oversight designed as an operating process
- A distinct and scoped identity for each agent
- Clear controls over the data each agent may access
- A remediation path for failed or partial writes
- An explicit process for exceptions

Start with a narrow use case, but build the controls needed for later scale. The majority of this foundation is widely accessible today. You secure true competitive advantage by mastering the complex pieces that remain.

Image generated with AI

About the Authors

Tjun Tang
Managing Director and Senior Partner
Hong Kong
Tang.Tjun@BCG.com

Dennis Trawnitschek
Former Group CTO, SCBX
Bangkok
dennis.trawnitschek@gmail.com

Luc Grimond
Managing Director and Senior Partner
Singapore
Grimond.Luc@bcg.com

Suttipong Kanakakorn
CEO, SCB Tech X
Bangkok
suttipong.kanakakorn@scbtechx.io

Srihari Chakarajan
Managing Director and Partner
Singapore
Srihari.Chakrarajan@bcg.com

Amir Alsbih
Partner and Associate Director, Data & Digital Platform
Munich
Alsbih.Amir@bcg.com

Disclaimer: The views expressed in this article are those of the authors and do not represent the official views, policies, or positions of the organizations with which the authors are affiliated. This article is intended to contribute to discussion on emerging approaches to AI in financial services and should not be construed as legal, regulatory, investment, technology, or professional advice.

This report was commissioned by **SCB X Public Company Limited.**

This document has been prepared in good faith on the basis of information available at the date of publication without any independent verification. The authors do not guarantee or make any representation or warranty as to the accuracy, reliability, completeness, or currency of the information in this document nor its usefulness in achieving any purpose. Readers are responsible for assessing the relevance and accuracy of the content of this document. It is unreasonable for any party to rely on this document for any purpose and the authors will not be liable for any loss, damage, cost, or expense incurred or arising by reason of any person using or relying on information in this document. To the fullest extent permitted by law, the authors shall have no liability whatsoever to any party, and any person using this document hereby waives any rights and claims it may have at any time against the authors with regard to the document. Receipt and review of this document shall be deemed agreement with and consideration for the foregoing.

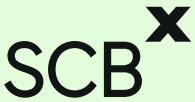
This document is based on primary qualitative and quantitative research executed by the authors. The authors do not provide legal, accounting, or tax advice. Parties are responsible for obtaining independent advice concerning these matters. This advice may affect the guidance in the document. Further, the authors have made no undertaking to update the document after the date hereof, notwithstanding that such information may become outdated or inaccurate. The authors do not provide fairness opinions or valuations of market transactions, and this document should not be relied on or construed as such. Further, any evaluations, projected market information, and conclusions contained in this document are based upon standard valuation methodologies, are not definitive forecasts, and are not guaranteed by the authors. The authors have used data from various sources and assumptions provided from other sources. The authors have not independently verified the data and assumptions from these sources used in these analyses. Changes in the underlying data or operating assumptions will clearly impact the analyses and conclusions.

This document is not intended to make or influence any recommendation and should not be construed as such by the reader or any other entity.

This document does not purport to represent the views of the companies mentioned in the document. Reference herein to any specific commercial product, process, or service by trade name, trademark, manufacturer, or otherwise, does not necessarily constitute or imply its endorsement, recommendation, or favoring by BCG.

About Boston Consulting Group

Boston Consulting Group bridges the gap between ambition and outcomes for the world's leading companies and organizations. We are built for this era of unprecedented change—bringing strategic clarity rooted in over 60 years of deep domain knowledge, combined with applied AI shaped by our practitioners. BCG works shoulder-to-shoulder with CEOs across industries and geographies to deliver transformative impact at scale: stronger returns, transferred capabilities, and change that sticks. For more information, visit bcg.com.



About SCBX Public Company Limited (SCBX)

SCBX is the mothership of the financial technology business group, comprising 13 subsidiary companies that operate across three key business pillars: Banking Business, Consumer and Digital Finance Business, and Platform and Technology Business. In addition, SCBX also focuses on Climate Technology, aspiring to become 'The Most Admired Regional Financial Technology Group'. The company conducts its business with flexibility and prudence in governance and risk management and has possesses the potential to compete equally in global competitions.

BCG | SCB X

Image generated with AI